IBM

Service-oriented modeling and architecture

How to identify, specify, and realize services for your SOA

Level: Intermediate

Ali Arsanjani, Ph.D. (arsanjan@us.ibm.com), Chief Architect, SOA and Web services Center of Excellence, IBM

09 Nov 2004

This article discusses the highlights of service-oriented modeling and architecture; the key activities that you need for the analysis and design required to build a Service-Oriented Architecture (SOA). The author stresses the importance of addressing the techniques required for the identification, specification and realization of *services*, their *flows and composition*, as well as the enterprise-scale *components* needed to realize and ensure the quality of services required of a SOA.

Introduction

There has been a lot of buzz and hype -- some factual, some not so well-founded -- surrounding the opportunities presented by Service-oriented Architectures (SOA) and its implementation as Web services. Analysts have predicted, pundits have professed, professors have lectured, companies have scurried to sell what they had, as SOA products -- often missing the point that SOA is not a product. It's about bridging the gap between business and IT through a set of business-aligned IT services using a set of design principles, patterns, and techniques.

ZDNet recently said, "Gartner predicts that by 2008, more than 60 percent of enterprises will use SOA as a "guiding principle" when creating mission-critical applications and processes."

A huge demand exists for the development and implementation of SOAs. So if SOA is not just about the products and standards that help realize it, for example through Web services, then what additional elements do you need to realize a SOA? *Service-oriented modeling* requires additional activities and artifacts that are not found in traditional object-oriented analysis and design (OOAD). The article "Elements of Service-oriented Analysis and Design" describes an initial set of reasons why you need more than OOAD. It also describes how business process management or enterprise architecture (EA) and OOAD are inadequate means of conducting analysis and design. Also, in the IBM Redbook entitled "Patterns: Service-Oriented Architecture and Web Services", I illustrate the major activities of this service-oriented modeling approach.

However, additional important considerations exist that you need to take into account. For one thing, current OOAD methods do not address the three key elements of a SOA: *services*, *flows*, and *components* realizing services. You also need to be able to explicitly address the techniques and processes required for the identification, specification and realization of services, their flows and composition, as well as the enterprise-scale components needed to realize and ensure the quality of services required.

Second, a paradigm shift needs to occur, in order to appreciate the distinct requirements of the two key roles in a SOA: the service provider and service consumer. Third, applications assumed to be built for one enterprise or line of business must now aspire to be used in a supply chain and be exposed to business partners who might compose, combine, and encapsulate them into new applications. This is the notion of the service ecosystem or service value-net.

This is a slight step up from just "distributed objects". It's about the value created through the network effect: for example, when parties leverage a combination of Amazon.com with Google's search services and combine them with eBay services to build their own hybrid solutions. Or when a travel agency reaches deep into an

airline reservation system and coordinates it with a rental car agency and hotel chain, updating their records and sending the itinerary to your electronic organizer. Whatever the application, you need much more than just good tools and standards to successfully create a SOA. You need some prescriptive steps to support your SOA life cycle; techniques for the analysis, design, and realization of services, flows, and components. Therefore, for anyone interested in enterprise application development, it's crucial to understand the detailed steps involved in service-oriented modeling and architecture.

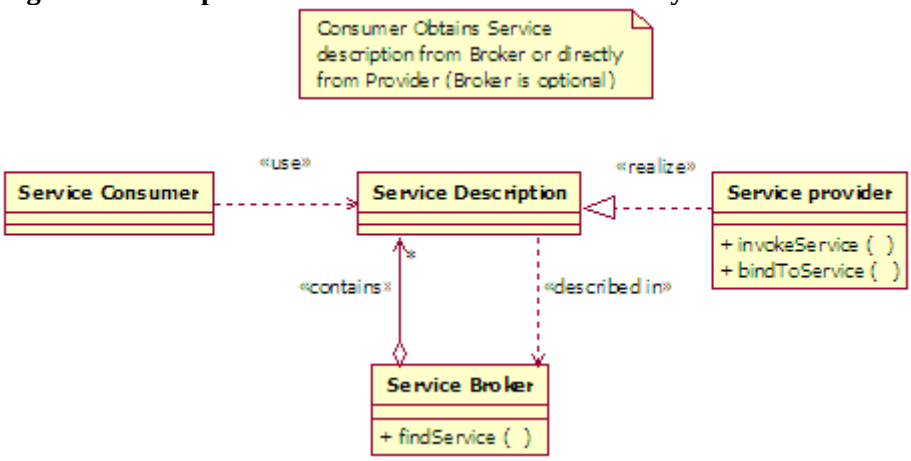
Before I describe these steps in detail, let's first understand what you are aiming to produce: *what is a SOA, and what does it look like?* After defining the notion and concepts behind a SOA, I'll describe the layers of a SOA and how you need to record key architectural decisions about each of those layers that help you in building a blueprint for the SOA that is right for your project, line of business, enterprise-wide effort, or value-chain that you are trying to integrate and come up with a set of services, flows, and components that implement the SOA.

Service-Oriented Architecture: A conceptual model

This concept is based on an architectural style that defines an interaction model between three primary parties: the service provider, who publishes a service description and provides the implementation for the service, a service consumer, who can either use the uniform resource identifier (URI) for the service description directly or can find the service description in a service registry and bind and invoke the service. The service broker provides and maintains the service registry, although nowadays public registries are not in vogue.

A meta-model showing these relationships is depicted in Figure 1 below.

Figure 1: Conceptual model of a SOA architectural style



The architectural style and principles

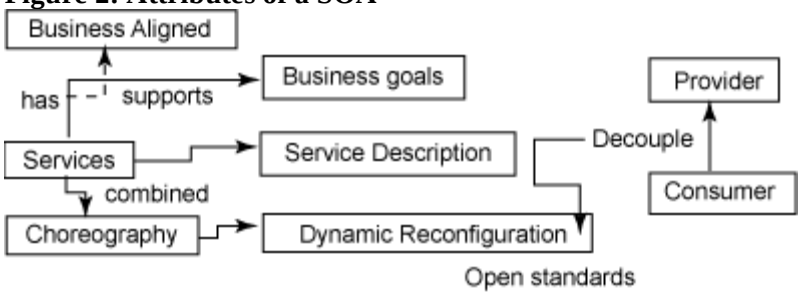
The architecture style defining a SOA describes a set of patterns and guidelines for creating *loosely coupled, business-aligned* services that, because of the separation of concerns between description, implementation, and binding, provide unprecedented flexibility in responsiveness to new business threats and opportunities.

A SOA is an enterprise-scale IT architecture for linking resources on demand. In a SOA, resources are made available to participants in a value net, enterprise, line of business (typically spanning multiple applications within an enterprise or across multiple enterprises). It consists of a set of business-aligned IT services that collectively fulfill an organization's business processes and goals. You can choreograph these services into

composite applications and invoke them through standard protocols, as shown in Figure 2 below.

A service is a software resource (discoverable) with an externalized service description. This service description is available for searching, binding, and invocation by a service consumer. The service provider realizes the service description implementation and also delivers the quality of service requirements to the service consumer. Services should ideally be governed by declarative policies and thus support a dynamically re-configurable architectural style.

Figure 2: Attributes of a SOA



Business agility is gained by IT systems that are flexible, primarily by separation of interface, implementation, and binding (protocols) offered by a SOA, allowing the deferral of the choice of *which* service provider *to opt for* at a given point in time *based on new business requirements*, (functional and non-functional (for example, performance, security, scalability, and so forth) requirements).

You can reuse the services across internal business units or across the value chains among business partners in a *fractal realization pattern*. Fractal realization refers to the ability of an architectural style to apply its patterns and the roles associated with the participants in its interaction model in a composite manner. You can apply it to one tier in an architecture and to multiple tiers across the enterprise architecture. Among projects, it can be between business units and business partners within a value chain in a uniform and conceptually scalable manner.

Context

In this article, I introduce the high-level activities of identification, specification and realization, and some artifacts of service-oriented modeling. Service-oriented modeling is a service-oriented analysis and design (SOAD) process for modeling, analyzing, designing, and producing a SOA that aligns with business analysis, processes, and goals.

I'll first take a look at what you intend to build, namely a SOA and its layers. Then I'll describe how to build the SOA by discussing the main activities and techniques that you need to create a SOA.

As an example, let's assume that you are working on a project and the objective is to migrate a portion of the banking line of business, which has a self-service accounting system, to a SOA.

In order to migrate to a SOA, you need some additional elements that go beyond service modeling. These include:

- Adoption and maturity models. Where is your enterprise at in the relative scale of maturity in the adoption of SOA and Web Services? Every different level of adoption has its own unique needs.
- Assessments. Do you have some pilots? Have you dabbled into Web services? How good is your resulting architecture? Should you keep going in the same direction? Will this scale to an enterprise SOA? Have you considered everything you need to consider?
- Strategy and planning activities. How do you plan to migrate to a SOA? What are the steps, tools, methods, technologies, standards, and training you need to take into account? What is your roadmap

and vision, and how do you get there? What's the plan?
- Governance. Should existing API or capability become a service? If not, which ones are eligible? Every service should be created with the intent to bring value to the business in some way. How do you manage this process without getting in the way?
- Implementation of best-practices. What are some tried and tested ways of implementing security, ensuring performance, compliance with standards for interoperability, designing for change?

In addition to identification, specification, and realization described in this article, the service-oriented modeling approach includes the techniques required for deployment, monitoring, management, and governance required to support the full SOA life cycle.

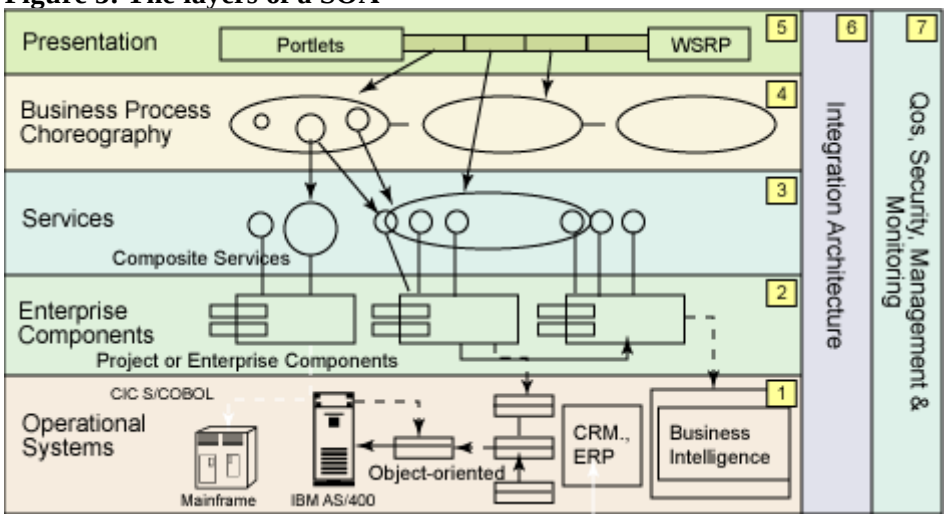
The above discussions on migration to SOA and the additional activities after realization deserve an article of their own, which I will get to in a subsequent column in this series. For now, let's assume that you scoped the project and decided where to focus on: a focal point for transformation of existing systems or services to a new set of systems and services has been defined. You can now start service-oriented modeling to build your service-oriented architecture.

An architectural template for a SOA

An abstract view of SOA depicts it as a partially layered architecture of composite services that align with business processes. Figure 3 depicts a representation of this type of architecture.

The relationship between services and components is that enterprise-scale components (large-grained enterprise or business line components) realize the services and are responsible for providing their functionality and maintaining their quality of service. Business process flows can be supported by a choreography of these exposed services into composite applications. An integration architecture supports the routing, mediation, and translation of these services, components, and flows using an Enterprise Service Bus (ESB). The deployed services must be monitored and managed for quality of service and adherence to non-functional requirements.

Figure 3: The layers of a SOA



For each of these layers, you must make design and architectural decisions. Therefore, to help document your SOA, you might want to create a document consisting of sections that correspond to each of the layers.

Here is a template for your SOA architecture document:

1. Scope <what area of the enterprise is this architecture for?>

2. Operational systems layer
    1. Packaged applications
    2. Custom applications
    3. Architectural decisions
3. Enterprise components layer
    1. Functional areas supported by this enterprise components
    2. <What business domains, goals and processes are supported by this enterprise components>
    3. Decisions regarding governance
        1. <Criteria by which something is elected as an enterprise components within this client organization>
    4. Architectural decisions
4. Services layer
    1. Categorized portfolio of services
    2. Architectural decisions
5. Business process and composition layer
    1. Business processes to be represented as choreographies
    2. Architectural decisions
        1. <Which processes need to be soft-wired into choreographies and which will be built into applications?>
6. Access or presentation layer
    1. <Document implications of Web services and SOA on this layer; if any. For example, use of portlets that invoke Web services at the user interface level and the implications on the functioning of that layer>
7. Integration layer
    1. <Include considerations of an ESB>
    1. <How are we going to ensure the service-level agreements (SLAs) and quality of service (QoS) required by clients of the services provided?>
    2. Security issues and decisions
    3. Performance issues and decisions
    4. Technology and standards limitations and decisions
    5. Monitoring and management of services
        1. Description and decisions

Now, let's describe each layer in greater detail and discuss the composition of each of these layers.

Layer 1: Operational systems layer. This consists of existing custom built applications, otherwise called *legacy* systems, including existing CRM and ERP packaged applications, and *older* object-oriented system implementations, as well as business intelligence applications. The composite layered architecture of an SOA can leverage existing systems and integrate them using service-oriented integration techniques.

Layer 2: Enterprise components layer. This is the layer of enterprise components that are responsible for realizing functionality and maintaining the QoS of the exposed services. These special components are a managed, governed set of enterprise assets that are funded at the enterprise or the business unit level. As enterprise-scale assets, they are responsible for ensuring conformance to SLAs through the application of architectural best practices. This layer typically uses container-based technologies such as application servers to implement the components, workload management, high-availability, and load balancing.

Layer 3: Services layer. The services the business chooses to fund and expose reside in this layer. They can be *discovered* or be statically bound and then invoked, or possibly, choreographed into a composite service. This service exposure layer also provides for the mechanism to take enterprise scale components, business unit specific components, and in some cases, project-specific components, and externalizes a subset of their interfaces in the form of service descriptions. Thus, the enterprise components provide service realization at

runtime using the functionality provided by their interfaces. The interfaces get exported out as service descriptions in this layer, where they are exposed for use. They can exist in isolation or as a composite service.

Level 4: Business process composition or choreography layer. Compositions and choreographies of services exposed in Layer 3 are defined in this layer. Services are bundled into a flow through orchestration or choreography, and thus act together as a single application. These applications support specific use cases and business processes. Here, visual flow composition tools, such as IBM® WebSphere® Business Integration Modeler or Websphere Application Developer Integration Edition, can be used for the design of application flow.

Layer 5: Access or presentation layer. Although this layer is usually out of scope for discussions around a SOA, it is gradually becoming more relevant. I depict it here because there is an increasing convergence of standards, such as Web Services for Remote Portlets Version 2.0 and other technologies, that seek to leverage Web services at the application interface or presentation level. You can think of it as a future layer that you need to take into account for future solutions. It is also important to note that SOA decouples the user interface from the components, and that you ultimately need to provide an end-to-end solution from an access channel to a service or composition of services.

Level 6: Integration (ESB). This layer enables the integration of services through the introduction of a reliable set of capabilities, such as intelligent routing, protocol mediation, and other transformation mechanisms, often described as the ESB (see Resources). Web Services Description Language (WSDL) specifies a binding, which implies a location where the service is provided. On the other hand, an ESB provides a location independent mechanism for integration.

Level 7: QoS. This layer provides the capabilities required to monitor, manage, and maintain QoS such as security, performance, and availability. This is a background process through sense-and-respond mechanisms and tools that monitor the health of SOA applications, including the all important standards implementations of WS-Management and other relevant protocols and standards that implement quality of service for a SOA.

How to approach service-oriented modeling and architecture

This section describes how to *combine* a top-down, business-driven approach with a bottom-up approach, leveraging legacy investments.

Service-oriented modeling approach provides modeling, analysis, design techniques, and activities to define the foundations of a SOA. It helps by defining the elements in each of the SOA layers and making critical architectural decisions at each level. It does so using a combination of a top-down, business-driven manner of service identification coupled with a stream of work that conducts service identification through leveraging legacy assets and systems.

In this way, high-level business process functionality is externalized for large-grained services. Smaller-grained services -- those that help realize the higher level of services -- are identified by examining the existing legacy functionality and deciding how to create adaptors and wrappers, or componentizing the legacy to externalize the desired functionality often locked within the system.

Finally, using goal-service modeling, you use a *cross-sectional* approach to cut down the sheer number of candidate services that might already be identified. A more judicious approach would be to first do top-down, then goal-service modeling, and finally bottom-up legacy analysis of existing assets. The message is: the faster you scope the project down to a manageable and realistic set, the sooner you can realize value by focusing on key services to expose with service descriptions that form the cornerstone of the SOA.

This combination of functional business aspirations and leveraging of existing investments in legacy systems

provide a potent solution to organizations that want to have quick wins and migrate their enterprise to a modern SOA. Consolidation of software applications through service-oriented integration thus becomes possible.

Service-oriented integration is an evolution of Enterprise Application Integration (EAI) in which proprietary connections are replaced with standards-based connections over an ESB notion that is location transparent and provides a flexible set of routing, mediation, and transformation capabilities.

Service-oriented modeling: The analysis and design of services

So far, I have set the stage by describing a SOA. I have also shown that to build a SOA, you need to make key architectural decisions about each layer in your SOA, and that your design must reflect a set of business-aligned services and decisions about how they will be composed into applications using choreography.

Unlike your comfortable world of objects, you need to take into account two perspectives in a SOA; that of the service consumer and service provider. The service broker is currently not mainstream and will be covered in a later venue.

The design strategy for a SOA does not start from the “bottom-up" as is often the case with a Web services-based approach. You must remember that SOA is more strategic and business-aligned. Web services are a tactical implementation of SOA. A number of important activities and decisions exist that influence not just integration architecture but enterprise and application architectures as well. They include the activities from the two key views of the consumer and provider described in Figure 4 below.

Figure 4: Activities of service-oriented modeling

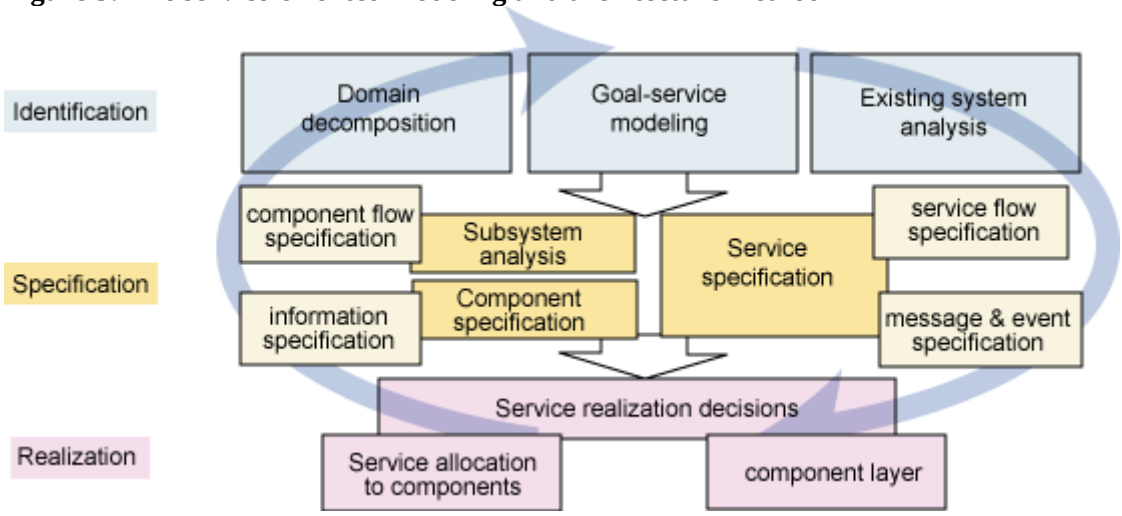
Role	Activities in this role				
Consumer view	Service identification	Service categorization	Service exposure decisions	Choreography or composition	Quality of service
Provider view	Component identification	Component specificaton	Service realization	Service management	Standards implementation
	Service allocations to components	Layering the SOA	Technical prototyping	Product selection	Architectural decisions (state, flow, dependencies)

Figure 4 shows the activities that are typically conducted by each of the roles of provider and consumer. Note that the provider’s activities are a superset of the consumer’s activities (for example, the provider would also be concerned with service identification, categorization, and so forth). In many cases, the differentiation of the roles comes from the fact that the consumers specify the services they want, often search for it, and once they are convinced of the match between the specification of the service they are looking for, and that provided by a service provider, they bind and invoke the service as needed. The provider, in turn, needs to publish the services they are willing to support; both in terms of functionality and most importantly in terms of the QoS that consumers will require. This implicit contract between consumer and provider might mature into an explicit contract in terms of SLAs; negotiated either electronically or through business and legal venues.

The activities described above can be depicted to flow within the service-oriented modeling and architecture

method, as shown in Figure 5 below.

Figure 5: The service-oriented modeling and architecture method



The process of service-oriented modeling and architecture consists of three general steps: identification, specification and realization of services, components and flows (typically, choreography of services).

Service identification

This process consists of a combination of top-down, bottom-up, and middle-out techniques of domain decomposition, existing asset analysis, and goal-service modeling. In the *top-down view*, a blueprint of business use cases provides the specification for business services. This top-down process is often referred to as *domain decomposition*, which consists of the decomposition of the business domain into its functional areas and subsystems, including its flow or process decomposition into processes, sub-processes, and high-level business use cases. These use cases often are very good candidates for business services exposed at the edge of the enterprise, or for those used within the boundaries of the enterprise across lines of business.

In the *bottom-up portion* of the process or *existing system analysis*, existing systems are analyzed and selected as viable candidates for providing lower cost solutions to the implementation of underlying service functionality that supports the business process. In this process, you analyze and leverage API's, transactions, and modules from legacy and packaged applications. In some cases, componentization of the legacy systems is needed to re-modularize the existing assets for supporting service functionality.

The middle-out view consists of *goal-service modeling* to validate and unearth other services not captured by either top-down or bottom-up service identification approaches. It ties services to goals and sub-goals, key performance indicators, and metrics.

Service classification or categorization

This activity is started when services have been identified. It is important to start service classification into a service hierarchy, reflecting the composite or fractal nature of services: services can and should be composed of finer-grained components and services. Classification helps determine composition and layering, as well as coordinates building of interdependent services based on the hierarchy. Also, it helps alleviate the service proliferation syndrome in which an increasing number of small-grained services get defined, designed, and deployed with very little governance, resulting in major performance, scalability, and management issues. More importantly, service proliferation fails to provide services, which are useful to the business, that allow for the economies of scale to be achieved.

Subsystem analysis

This activity takes the subsystems found above during domain decomposition and specifies the interdependencies and flow between the subsystems. It also puts the use cases identified during domain

decomposition as exposed services on the subsystem interface. The analysis of the subsystem consists of creating object models to represent the internal workings and designs of the containing subsystems that will expose the services and realize them. The design construct of “subsystem" will then be realized as an implementation construct of a large-grained component realizing the services in the following activity.

Component specification

In the next major activity, the details of the component that implement the services are specified:

- Data
- Rules
- Services
- Configurable profile
- Variations

Messaging and events specifications and management definition occur at this step.

Service allocation

Service allocation consists of assigning services to the subsystems that have been identified so far. These subsystems have enterprise components that realize their published functionality. Often you make the simplifying assumption that the subsystem has a one-to-one correspondence with the enterprise components. *Structuring components* occurs when you use patterns to construct enterprise components with a combination of:

- Mediators
- Façade
- Rule objects
- Configurable profiles
- Factories

Service allocation also consists of assigning the services and the components that realize them to the layers in your SOA. Allocation of components and services to layers in the SOA is a key task that require the documentation and resolution of key architectural decisions that relate not only to the application architecture but to the technical operational architecture designed and used to support the SOA realization at runtime.

Service realization

This step recognizes that the software that realizes a given service must be selected or custom built. Other options that are available include integration, transformation, subscription and outsourcing of parts of the functionality using Web services. In this step you make the decision as to which legacy system module will be used to realize a given service and which services will be built from the “ground-up". Other realization decisions for services other than business functionality include: security, management and monitoring of services.

In reality, projects tend to capitalize on any amount of parallel efforts to meet closing windows of opportunity. Therefore, I recommend conducting three streams in parallel.

Top-down domain decomposition (process modeling and decomposition, variation-oriented analysis, policy and business rules analysis, and domain specific behavior modeling (using grammars and diagrams)) is conducted in parallel with a bottom-up analysis of existing legacy assets that are candidates for componentization (modularization) and service exposure. To catch the business intent behind the project and to align services with this business intent, goal-service modeling is conducted.

Conclusion

In this article, I started with the fundamentals of a service-oriented architecture, its layers, and the associated types of architectural decisions. Then, I described the activities in a method for service-oriented modeling and the importance of activities from the service consumer and provider perspectives (service broker will be dealt with in a later article). This method provides specific guidance on the analysis and design activities for determining the three fundamental aspects of a service-oriented architecture: services, flows, and components that realize the services. I also described a template you can use for your architectural decisions in each of the layers of the SOA.

I noted that for service identification, it is important to combine the three approaches of top-down, bottom-up, and cross-sectional, goal-model analysis. I then looked at the main activities of specification and realization of services.

In the next column in this series, I will apply the method to the banking domain of account management and describe each step with an example. In addition to identification, specification, and realization, I will also discuss the remaining activities of the service-oriented modeling approach that includes deployment, monitoring, management, and governance required to support the full SOA life cycle.

Acknowledgments

The author would like to acknowledge the valued input and feedback of the following esteemed colleagues and friends (no particular order): Luba Cherbakov, Kerrie Holley, George Galambos, Sugandh Mehta, David Janson, Shankar Kalyana, Ed Calunzinski, Abdul Allam, Peter Holm, Krishnan Ramachandran, Jenny Ang, Jonathan Adams, Sunil Dube, Ralph Wiest, Olaf Zimmerman, Emily Plachy, Kathy Yglesias-Reece, and David Mott.

Resources

- Read the article “Elements of Service-oriented Analysis and Design", (developerWorks, June 2004) for more information on this interdisciplinary modeling approach for SOA projects.
- Read about "Patterns: Service-oriented Architecture and Web Services" Redbook, SG24-6303-00, April 2004, Endrei M., et al.
- Visit WebServices.Org™ for more information on the *Goal-oriented approach to enterprise component identification and specification*, Communications of the ACM, Oct 2002, K. Levi, A. Arsanjani.
- Get the *Externalizing Component Manners to Achieve Greater Maintainability through a Highly Re-Configurable Architectural Style* article by Ali Arsanjani, James J. Alpigini, and Hussein Zedan. Proceedings of the ICSM: 628-. IEEE Press 2002.
- Go to Ali Arsanjani's Patterns and Best-practices Web site for more information on The Enterprise Component Pattern, proceedings of pattern languages of programming 2000.
- Check out the "Patterns: Implementing an SOA with the Enterprise Service Bus" Redbook, SG24-6346-00, August 2004, Martin Keen, Susan Bishop, Alan Hopkins, Sven Milinski, Chris Nott, Rick Robinson, Jonathan Adams, Paul Verschueren.
- Access Web services knowledge, tools, and skills with Speed-start Web services, which offers the latest Java-based software development tools and middleware from IBM (trial editions), plus online tutorials and articles, and an online technical forum.
- Get involved in the developerWorks community by participating in developerWorks blogs.
- Browse for books on these and other technical topics.

- Want more? The developerWorks SOA and Web services zone hosts hundreds of informative articles and introductory, intermediate, and advanced tutorials on how to develop Web services applications.

About the author

Dr. Ali Arsanjani is a Senior Technical Staff Member who is Chief Architect of the SOA and Web Services Center of Excellence in IBM Global Services. He has 21 years experience in the IT industry, designing and delivering distributed software architectures for larger systems. His research interests and publications include software design patterns, software architecture, component-based and service-oriented architectures, and grammar-oriented object design. He specializes in building dynamically reconfigurable software systems. You can contact him at arsanjan@us.ibm.com.